使用Python进行Dask并行编程

大数据并行计算

吴奔 wuben@ruc.edu.cn 2023秋

使用Anaconda创建python环境

- 更换conda源，修改.condarc文件（它的位置在cd ~/）：

```
channels:
  - defaults
show_channel_urls: true
default_channels:
  - http://mirrors.aliyun.com/anaconda/pkgs/main
  - http://mirrors.aliyun.com/anaconda/pkgs/r
  - http://mirrors.aliyun.com/anaconda/pkgs/msys2
custom_channels:
  conda-forge: http://mirrors.aliyun.com/anaconda/cloud
  msys2: http://mirrors.aliyun.com/anaconda/cloud
  bioconda: http://mirrors.aliyun.com/anaconda/cloud
  menpo: http://mirrors.aliyun.com/anaconda/cloud
  pytorch: http://mirrors.aliyun.com/anaconda/cloud
  simpleitk: http://mirrors.aliyun.com/anaconda/cloud
```

使用Anaconda创建python环境

- 清除索引缓存：
  - conda clean -i
- 更新conda：
  - conda update conda

启动Jupyter Notebook

- 启动Jupyter Notebook（在服务器）：
  - jupyter notebook --allow-root --no-browser

```
[I 21:17:20.504 NotebookApp] The Jupyter Notebook is running at:
[I 21:17:20.504 NotebookApp] http://localhost:8888/?token=e53c5a72ab3bee0eda1849
5761fa87474ff4f4101d0c1a92
[I 21:17:20.504 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 21:17:20.504 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=e53c5a72ab3bee0eda18495761fa87474ff4f4101d0
c1a92
```

- 在本地输入：
  - ssh -L 8888:localhost:8888 <username>@XX.XX.XX.XX
- 在浏览器中输入http://localhost:8888/就可以远程访问Jupyter Notebook

Dask简介

- Dask是一个并行计算库，比Spark更轻便；
- Dask侧重与其他框架（如NumPy、Pandas、Scikit-learn）相结合，更方便进行并行计算。
- Dask由两部分组成：
  - 针对计算优化的动态任务调度；
  - “大数据”集合，像并行数组Array、数据框DataFrame和列表Bag一样，将通用接口扩展到大于内存或分布式环境。

Dask的主要优点

- 熟悉：提供并行的NumPy和Pandas Dataframe；
- 灵活：提供任务计划界面，实现更多自定义工作负载并与其他项目集成；
- 本土化：在纯Python中启用分布式计算并可以访问PyData堆栈；
- 快速：以低开销、低延迟和快速数值算法执行所需的最少序列化操作；
- 扩大规模：在具有1000个核心的集群上弹性运行；
- 缩小：在单个过程中轻松设置并在笔记本电脑上运行；
- 响应式：在设计时考虑了交互式计算，可提供快速反馈和诊断功能。

安装Dask

- 代码默认在Jupyter Notebook中执行；
- 安装Dask全部功能：
  - !pip install “dask[complete]"
- 也可以根据需要安装Dask部分模块。

- Dask存在三种最基本的数据结构：Array、DataFrame和Bag。

Dask对Array的操作

- Dask库的Array可以看作NumPy数组的拓展；
- 创建NumPy数组：
  - import numpy as np
  - a = np.random.rand(10000)
  - print(a.shape, a.dtype)
  - print(a.sum())
  - print(a.mean())

Dask对Array的操作

- 从NumPy数组创建Dask数组：
  - 导入模块dask.array作为da使用Dask数组；
  - 其中from_array函数从NumPy数组构造Dask数组；
  - Chunks表示生成的Dask数组中每个块包含的元素个数；
  - import dask.array as da
  - a_dask = da.from_array(a, chunks = len(a)//4)
  - a_dask.chunks

Dask对Array的操作

- NumPy数组分块求和：
  - 利用循环分块求和（如果可能也可以并行执行）；
  - n_chunks = 4
  - chunk_size = len(a)//n_chunks
  - result = 0
  - for k in range(n_chunks):
    - offset = k*chunk_size
    - a_chunk = a[offset:offset + chunk_size]
    - result += a_chunk.sum()
  - print(result)

Dask对Array的操作

- Dask数组分块求和：
  - 对sum调用会产生一个未计算的Dask对象；
  - a_dask = da.from_array(a, chunks = len(a)//n_chunks)
  - result = a_dask.sum()
  - result
  - 调用compute()强制计算sum，visualize()查看相关任务图（conda install python-graphviz）；
  - print(result.compute())
  - result.visualize(rankdir = 'LR')

Dask对Array的操作

- Dask数组分块求和：
  - 对sum调用会产生一个未计算的Dask对象；
  - a_dask = da.from_array(a, chunks = len(a)//n_chunks)
  - result = a_dask.sum()
  - result
  - 调用compute()强制计算sum，visualize()查看相关任务图（conda install python-graphviz）；
  - print(result.compute())
  - result.visualize(rankdir = 'LR')

Dask对Array的操作

- Dask与NumPy共享许多属性与方法：
  - print(a_dask.shape, a_dask.dtype)
  - a_dask.mean().compute()
  - a_dask.max().compute()
  - a_dask.reshape(2500,4).compute()

Dask对DataFrame的操作

- Dask DataFrame是Pandas DataFrame的延时版本；
- head()和tail()不需要调用compute()：
  - import dask.dataframe as dd
  - df = dd.read_csv('~/Data/2014-*.csv')
  - df.head()
  - df2 = df[df.y == 'a'].x + 1
  - df2.compute()

Dask对Bag的操作

- Bag可以对集合及Python对象进行map、filter、fold、groupby等操作；
- Bag的好处：
  - 并行：数据被拆分，允许多个内核或机器并行执行；
  - 迭代：数据处理延迟，即使在单个分区内的单个计算机上，也可以执行大于本地内存的数据。
- Dask Bag是一种方便的、异构的、类似列表的数据结构，可以处理非结构化的数据。

Dask对Bag的操作

- from_sequence函数将嵌套的容器转换为Dask Bag：
  - nested_containers = [[0,1,2,3],{}, [6.5,3.14],'Python', {'version':3},' ‘]
  - import dask.bag as db
  - the_bag = db.from_sequence(nested_containers)
  - the_bag.count().compute()
  - the_bag.any().compute(), the_bag.all().compute()

Dask对Bag的操作

- Dask包被设计用于处理凌乱或非结构化的文件，通常是原始ASCII或其他文本文件；
- read_text()逐行读入：
  - import dask.bag as db
  - zen = db.read_text('~/Data/zen.txt')
  - taken = zen.take(1)
  - type(taken)
  - taken
  - zen.take(3)

glob expression

- glob是由普通字符和/或通配符组成的字符串，用于匹配文件路径。可以利用一个或多个glob在文件系统中定位文件；
- 使用glob patterns中的通配符*来匹配taxi目录中以.csv结尾的所有文件名：
  - 斜杠字符是目录分隔符，星号是匹配0或多个字符的通配符；
  - import dask.dataframe as dd
  - df = dd.read_csv('~/Data/example/*.csv', assume_missing = True)

glob expression

- 修改路径：
  - import os
  - os.getcwd()
  - os.chdir('/root/Data/example/')
- 查看目录：
  - %ls

glob expression

- glob.glob函数返回字符串列表，这些字符串使用glob模式来匹配工作目录中的文件：
  - import glob
  - glob.glob('*.txt')
  - glob.glob('b*.txt')
- 问号通配符只匹配一个字符：
  - glob.glob(‘b0?.txt')
- 方括号用于提供字符值的范围：
  - glob.glob('?0[1-6].txt')
  - glob.glob('??[1-6].txt')

map和filter

- map会根据提供的函数对指定序列做映射：
  - def squared(x): return x ** 2
  - squares = map(squared, [1,2,3,4,5,6])
  - squares
  - squares = list(squares)
  - squares

map和filter

- filter对指定的序列过滤操作：
  - def is_even(x): return x % 2 == 0
  - evens = filter(is_even, [1,2,3,4,5,6])
  - list(evens)
  - even_squares = filter(is_even, squares)
  - list(even_squares)

map和filter

- 使用dask.bag.map和dask.bag.filter：
  - import dask.bag as db
  - numbers = db.from_sequence([1,2,3,4,5,6])
  - squares = numbers.map(squared)
  - squares.compute()
  - evens = numbers.filter(is_even)
  - evens.compute()
  - numbers.map(squared).filter(is_even).compute()

在大数据集上训练

- Dask-ML包实现了可以在大于计算机内存的Dask数组或数据帧上进行机器学习训练；
- 安装和加载：
  - !pip install dask-ml
  - import dask.array as da
  - import dask.delayed
  - from sklearn.datasets import make_blobs
  - import numpy as np

创建随机数据集

- 使用scikit-learn在本地创建一个小型（随机）数据集：
  - n_features = 20
  - n_centers = 50
  - X_small, y_small = make_blobs(n_samples=1000, centers=n_centers, n_features=n_features, random_state=0)
  - centers = np.zeros((n_centers, n_features))
  - for i in range(n_centers): centers[i] = X_small[y_small == i].mean(0)

生成数据集

- 使用dask.delayed，在工作节点上生成实际的数据集：
  - n_samples_per_block = 200000
  - n_blocks = 500
  - delayeds = [dask.delayed(make_blobs)(n_samples=n_samples_per_block, centers=centers, n_features=n_features, random_state=i)[0] for i in range(n_blocks)]
  - arrays = [da.from_delayed(obj, shape=(n_samples_per_block, n_features), dtype=X_small.dtype) for obj in delayeds]
  - X = da.concatenate(arrays)
  - X

K-means计算

- K-means代码（跑不了，未解决）：
  - from dask_ml.cluster import KMeans
  - clf = KMeans(init_max_iter=3, oversampling_factor=10)
  - clf.fit(X)
  - clf.labels_
  - clf.labels_[:10].compute()

使用Dask可视化示例

- 创建h5py.Dataset（没找到这个数据，不知道在写什么）：
  - import h5py
  - from glob import glob
  - import os
  - filenames = sorted(glob(os.path.join('data', 'weather-big', ‘*.hdf5')))
  - dsets = [h5py.File(filename, mode = 'r')['/t2m'] for filename in filenames]

并行及分布式机器学习

- 生成随机数据：
  - from sklearn.datasets import make_classification
  - X, y = make_classification(n_samples=10000, n_features=4, random_state=0)
- 支持向量机：
  - from sklearn.svm import SVC
  - estimator = SVC(random_state=0)
  - est = estimator.fit(X,y)
  - estimator.score(X,y)

分布式部署示例

- Dask分布式有三种角色：
  - 主节点（scheduler）、工作节点（worker）和客户端（client）；
  - client提交task给scheduler，scheduler对提交的task按照一定的策略分发给worker，worker进行实际的计算、数据存储。
- Dask内部自动实现了分布式调度，无须用户自行编写复杂的调度逻辑和程序。

分布式部署示例

- 在终端输入dask-scheduler启动主节点；
- 伪分布式部署（启动以后就卡住了）： dask-worker 127.0.0.1:8786